WHITE PAPER

Scalable Computing: Why and How

J. Duato*, F. Silla
Technical University of Valencia, *Simula Labs

B. Holden, P. Miranda, J. Underhill, M. Cavalli
HyperTransport Consortium

S. Yalamanchili
Georgia Institute of Technology

U. Brüning, H. Fröning
University of Heidelberg

Executive Summary

HyperTransport 3.10 is the best open standard communication technology for chip-to-chip interconnects. However, its extraordinary features are of little help when building mid- and large-scale systems because it is unable to natively scale beyond 8 computing nodes. Therefore, it must be complemented by other interconnect technologies.

The HyperTransport Consortium has intensively stimulated discussions among its high-level members in order to overcome those shortcomings. The result is the High Node Count HyperTransport Specification, which defines protocol extensions to the HyperTransport I/O Link Specification Rev. 3.10 that enable HyperTransport to natively support high numbers of computing nodes, typical of large scale server clustering and High Performance Computing (HPC) applications. This extension has been carefully designed in such a way that it extends the maximum number of connected devices to a number large enough to support current and future scalability requirements, while preserving the excellent features that made HyperTransport successful and keeping full backward compatibility with it.

Introduction

HyperTransport 3.10 [5] (hereafter referred to as HT3.10 or simply as HT) is currently the lowest latency, highest bandwidth openly licensed standard communication technology for chip-to-chip and board-to-board interconnects. This performance leadership was achieved by:

- Minimizing packet protocol overhead
- Adopting a clock-forwarding scheme that eliminates clock recovery overhead
- Eliminating control and command signals required by other communication standards
- Reducing crosstalk and electromagnetic interference

HyperTransport was devised as an efficient replacement of the traditional system bus and it has become the interconnect of choice for on-board communications. Furthermore, HT Rev. 3.x specifications (HT3) significantly extended the scope of HT Rev. 2.0 by providing support for

chassis-to-chassis - i.e. short-haul system-to-system interconnects for rack-mounted server clusters - and backplane implementations. This is achieved through the AC mode and the link-splitting features. The former supports links up to 1m (3 feet) in length at full speed and the latter increases the number of HyperTransport links in and out of a device without having to increase the number of pins. Hot-plugging - also introduced with HT3 - helps to enhance HyperTransport's dynamic expansion capabilities and to improve system availability in HT-based servers and storage systems.

At link level, HT uses a lean packet protocol that carries significantly less overhead than other interconnect technologies. PCI Express, for instance, requires 2 bytes for framing and 20 percent 8b10b encoding overhead for the physical layer, 8 additional bytes for the data link layer and 12 or 16 extra bytes for the transaction layer. By contrast, HT requires no overhead for the physical layer and only 8 to 16 header bytes for the transaction/data link layer.

The characteristic that most uniquely distinguishes HyperTransport from other interconnection technologies in the market, however, is its being processor-native - i.e. integrated in the processor chip, as in the case of various AMD CPUs, as well as a number of specialty processors and SoCs from Bay Microsystems, Broadcom, NetLogic Microsystems, PMC-Sierra, Raza Microelectronics, and Tarari.

Traditionally, CPU-to-peripheral communication has been accomplished by going through a north bridge controller (competing with main memory accesses) and reaching the destination peripheral device via one of various communication standards, like PCI or PCI Express. The peripheral device would then communicate with the CPU by following the same path in reverse. By virtue of its processor-native support, HT processor-to-peripheral communication is speeded up because it takes place on a daisy-chained, direct point-to-point link in which intermediate control functions such as north bridge controllers - with their intrinsic overhead penalties - are eliminated. Additionally, no protocol translation is required. The result of such architectural innovation is that HyperTransport collectively combines interconnect integration with high bandwidth, low latency, and low implementation cost.

HyperTransport's processor-native feature has been demonstrated to be so successful in reducing communication latency that other microprocessor manufacturers, like Intel Corporation, have modified the way their processors communicate by including this feature in their own processors. To do so, Intel developed a new proprietary interconnect technology, called Quick Path Interconnect (QPI). The first Intel processors featuring this new interconnect technology were recently introduced to market.

Thanks to its powerful features, HyperTransport has the potential to weave off-the-shelf CPU subsystems and servers into highly scalable system fabrics and clusters. Examples are the 4-way and 8-way CPU architectures proposed by AMD and shown in Figures 1 and 2. Opteron chips in Figure 1 are natively interconnected via 3 coherent HT links per processor (chips may use an additional HT link for connecting to I/O, if required). By considering that more sockets could be directly connected via HT3's link-splitting feature, and that they could be populated with 8-core Opteron processors per AMD's product roadmap, these systems may straightforwardly become 64-way systems with today's HT specifications (Figure 2).

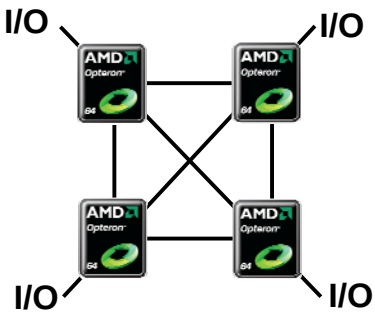

Figure 1: 4-way Opteron system. Processors are natively interconnected by HT

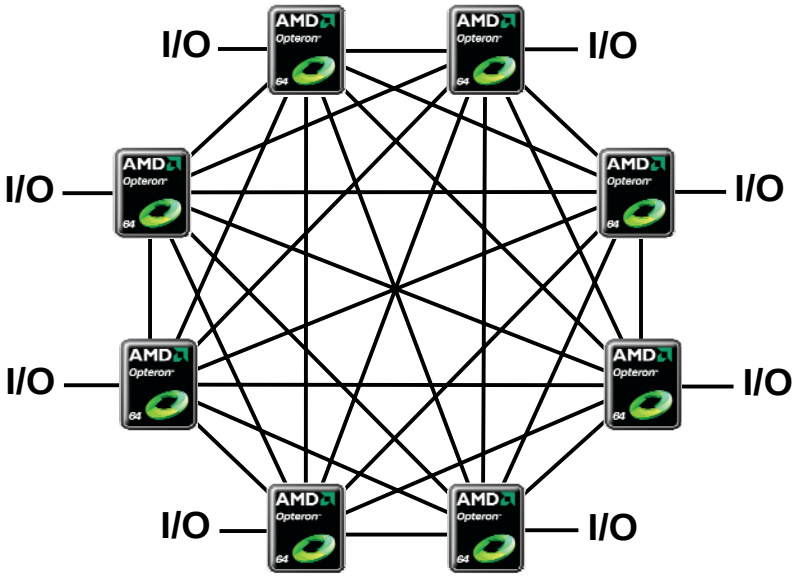

Figure 2: 8-way Opteron system. All-to-all connections are allowed by HT's link-splitting

Looking into the future, the necessity for higher performance and higher scalability solutions should make us - high-tech purveyors - alert that future High Performance Computing (HPC) platforms will have to support significantly greater scalability. This is to say that, if systems with tens of thousand of processing nodes continue to represent the elite play - i.e. limited volume opportunity with not necessarily limited profits - mid-scale systems with several hundreds of processors should progressively become common place. A growing market sector that HyperTransport Rev. 3.10 - by itself and with its present capabilities - will be increasingly unable to compete for and capitalize on.

The HyperTransport Consortium, aware of these limitations, has stimulated an extension to HT3.10 in order to overcome those shortcomings. As a result, the High Node Count HyperTransport Specification - born from the contribution of HyperTransport Consortium's high-level commercial and academic members - was recently released by the Consortium as an extension to the

HyperTransport 3.10 Link specification and providing the means for HyperTransport to support large systems. This paper presents such an extension to HT. To do so, the next section presents the market trends that motivated such an extension. Then, the subsequent section introduces the context for the extension by defining the system model to use. Next, the need for a HT protocol extension is discussed. After that, some of the considerations taken into account inside the HyperTransport Consortium during the development of the new specification are summarized, followed by a brief presentation of the new High Node Count HyperTransport Specification, also explaining why some of its features have been devised that way. Next section provides some insights on how to implement these extensions in the current technology arena. Finally, some conclusions are drawn.

HyperTransport Limitations

As described above, HyperTransport offers some degree of scalability latitude by virtue of its link-splitting, AC-mode, and hot-plugging capability, which could enable the implementation of efficient network topologies like 3D meshes or tori. However, such network topologies, when scaling to large sizes, require routing capabilities beyond current HyperTransport ones. Specifically, HT lacks support for the following:

- Global device addressability beyond 32 HT devices, required for medium and large size clusters of processing nodes
- Efficient routing in scalable network topologies
- Scalable congestion management mechanisms
- Dynamic reconfiguration of routing information after hot plug/swap/removal of components - i.e. no automatic finding of better routing paths after changing system topology

As a direct result of HT's scalability shortcomings, HPC vendors have no choice but to complement HyperTransport with other interconnect technologies. Examples are Sun Microsystems, the extinct Newisys, and Cray.

In the first case, Opteron boxes are interconnected via Gigabit Ethernet or InfiniBand, providing a non-shared memory system composed of several independent computers that communicate via some kind of message-passing protocols. Thus, the system cannot be viewed as a single large-scale system, but as an aggregation of small systems, between which communication takes place explicitly. This configuration is similar to the one traditionally used in clusters and PC farms based on Ethernet intra-system interconnect backbones, which can be further performance-accelerated by means of HT-enabled network interface Cards (NICs). However, such kind of communication model is burdened by the latency penalty introduced by the process of creating by software the messages that enable inter-processor communications. This latency penalty usually includes one or more system calls at the source and destination ends of the communication links, noticeably lowering performance. Additionally, peripheral devices cannot be easily shared among processors.

In the case of Cray's XT4 and XT5 supercomputers [4], HyperTransport provides a 6.4 GB/s direct connection between the Opteron processors and Cray's SeaStar interconnect backbone. The SeaStar interconnect is based on Cray's SeaStar2 chips and implements a proprietary protocol to directly connect up to 30,000 processing nodes in a 3D torus topology. With this approach, the cost and complexity of external switches is entirely removed and systems can be easily scaled in field. Figure 3 shows the profile of Cray's interconnect, with the AMD Opteron processors connected to the SeaStar chips via HT links (green pipes) and the SeaStar chip linking each processing node to all others via proprietary links and protocol (orange pipes). It is important to note that in addition to

being able to differentiate from competitors, the main reason that compels companies like Cray to use proprietary interconnects is not necessarily to attain higher bandwidth - i.e. HT3.10's 25.6 GB/s (16-bit) bandwidth is much greater than the 9.6 GB/s required by Cray - but highly likely to compensate for HyperTransport's inability to scale up to such large system requirements.

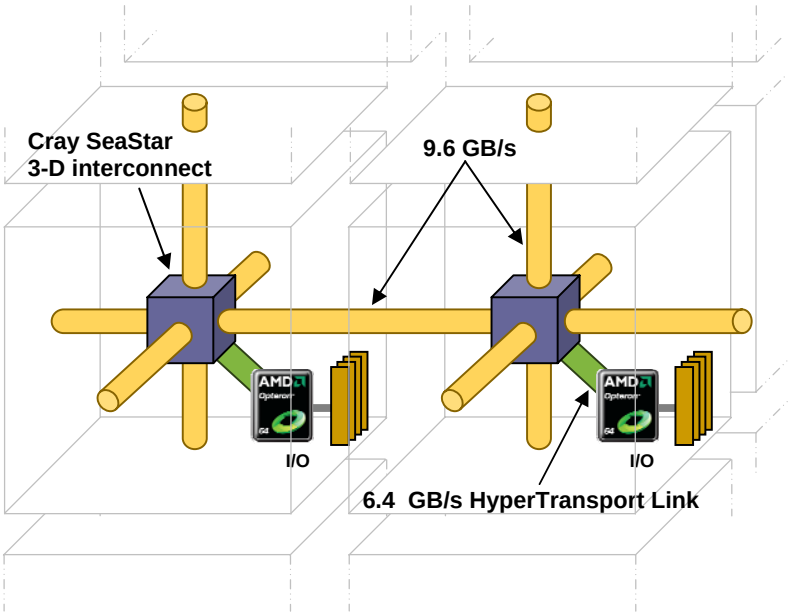

Figure 3: Cray XT4 scalable architecture. HyperTransport is used to connect Opteron chips to the proprietary interconnects

With its proprietary Horus chip [7], Newisys proposed a different approach, which extends HT's basic functionality and enables Symmetric Multi-Processing topologies of up to 32 AMD Opteron chips (32-way) with full cache coherence support. Horus chips appeared to AMD CPUs as CPUs themselves and provided special routing for data and command packets, as well as local cache and hidden directory scheme to significantly enhance the performance of cache coherence protocols. The resulting system was a single, coherent shared-memory machine that supported implicit communication without the use of system calls, thereby significantly accelerating communication among processors. Unfortunately, scalability of Horus-based systems was limited to only 32 host nodes likely due to the unsolved inability of cache coherent node clusters to scale efficiently and, in addition, to the intrinsic inability of effective scaling of AMD's proprietary cache coherence protocol.

In summary, HyperTransport is an excellent interconnection technology that provides the highest bandwidth and lowest latency. However, HyperTransport's benefits are primarily confined to host-to-host and host-to-I/O subsystems within the realm of a single motherboard. Even with the introduction of the AC mode in HT3, the extraordinary features of HyperTransport are of little help when building large systems because HT is unable to natively scale as required by mid- and large-scale HPC applications and, therefore, must be complemented by other interconnect technologies. Note that this inability is not due to insufficient bandwidth or connectivity. In fact, as Figure 2 shows, Opteron processors may have up to 8 HyperTransport links (by using the link splitting feature), and therefore, efficient network topologies, like 3-D meshes or tori can be implemented. However, these topologies

would require extending current HyperTransport scalability characteristics. This was even stated by AMD several years ago [1].

System Model and Definitions

Enhancing HT to natively support a large number of processors brings the opportunity to define a new system architecture that is scalable, flexible, and simplifies application development, all of this with minimal additional cost. As HyperTransport is a shared-memory oriented protocol, its enhancement would naturally provide a shared-memory system. However, it is well known that large-scale cache-coherent shared-memory systems have never been feasible. Large systems are message-passing flavoured, instead.

Devising a message-passing HT for high node count systems would deliver little improvement over current large installations, which are already message-passing oriented. Moreover, message-passing application programmers interfaces (APIs), like MPI, may not be enough in the multicore era [3]. Additionally, other programming models, like PGAS [8] are starting to play an important role. On one hand, their performance can equal that of MPI codes and, for most humans, they are much easier to learn [2]. Also, PGAS is not less scalable than MPI and permits sharing, whereas MPI rules it out [9]. On the other hand, PGAS implements a one-sided communication model (faster than two-sided), where caching is not required and the programmer makes local copies and manages their consistency. Because of this, no cache coherence protocol is needed, except between the network interface and the processes in a node. Additionally, a one-sided put/get message can be handled directly by a network interface with RDMA support, avoiding interrupting the CPU or storing data from it.

Because of all the benefits that a global address space delivers, the Consortium decided to allow the enhanced HyperTransport to naturally provide what it provided before: a shared-memory system. Note that now this system model may be efficiently supported because cache coherence is not maintained by hardware, and is not enforced for every memory access. And it is more efficient than other PGAS implementations because HT interfaces are directly attached to the processors.

Thus, the system model in mind was based on a large number of HyperTransport devices that use the HT protocol to perform memory transactions. This will be referred to as *High Node Count Network*. We may think of this network as a network of processors, each of them with its local memory and I/O, as depicted in Figure 4. The way such processors are physically interconnected is not relevant at this point of the discussion.

Moreover, in order to devise a general and architecturally independent system model, we should define the concept of *Nest*. A nest is defined as each of the components of the High Node Count Network. A nest may be something as simple as a single CPU or something much more complex, as a motherboard containing four CPU devices, each of them containing four processor cores. Basically, the term *Nest* refers to a network-addressable entity. In Figure 4, each of the processors depicted would be a nest. Hereafter, we will use the term *Nest* instead of CPU, processor, or motherboard.

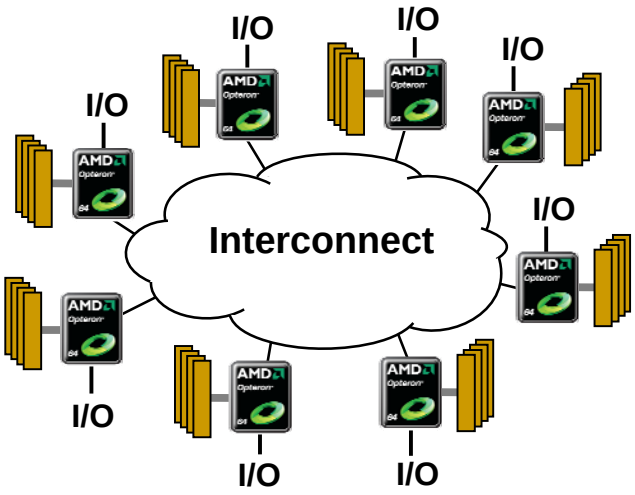

Figure 4: System model assumed in the High Node Count HyperTransport Specification

As one of the goals is providing HyperTransport with the capability of addressing a large number of devices, in the system model described above each nest will be assigned a *NestID* that unambiguously identifies it in the High Node Count Network. Note that a NestID is a network-style address and not a memory-style address. Moreover, a protocol that sets up the identifier of each nest during system initialization is required. Such a protocol, or a variant of it, may also be required after hot-plugging of components. The definition of these protocols is outside the scope of this discussion as far as a unique system-wide NestID is provided for each nest in the system.

In the system model proposed in Figure 4 - a physically distributed logically shared-memory system - each nest has access to its local memory via conventional memory buses and has access to memory belonging to other nests via HT packet exchanges. In these exchanges, the nest that sources the request must include its NestID in the packet as well as the NestID of the destination of the request, i.e., requests will carry a *SrcNest* and a *DestNest*. Once the request reaches its destination, that nest will use the SrcNest as the destination identifier for the response packet.

Need for Protocol Extensions

At first glance, it may seem that the considered distributed shared-memory model is completely compatible with current HT3 specification, as current HT requests that specify an address use 40-bit addresses with an optional 24-bit address extension and therefore, the upper part of the 64-bit global memory addresses could be used as the destination identifier. However, despite providing support for 64-bit addresses, the HT3 protocol does not provide the functionality required for a number of reasons:

1. Requests targeted to another nest may use the current address extension to identify the destination nest. However, the address extension can only be used for commands that include an address. Therefore, responses back to the source nest - which would also require a destination identifier to be appropriately routed inside the interconnect - cannot use an address extension as currently defined because current packet responses do not include an address field. Thus, responses could not be returned to the source nest. Similarly, some HT commands, like Flush, may be directed to remote nests. These commands do not include a

40-bit address in their packet and, therefore, they are not eligible to be extended by an address extension according to current HT specifications. However, when targeting these commands to a remote nest, a DestNest identifier is required in order to forward the packet to the right destination. Therefore, the address extension, as defined in the HT specification, does not provide the required support.

2. Once the target nest has accessed its local memory as the result of a remote request, it needs to know where to return the corresponding response. To accomplish this, it is necessary to include a SrcNest identifier in the requests, so that target nests use it as the destination identifier in the response. However, the source identifier extension as currently defined in the HT specifications does not allow this feature because it only includes a 16-bit address in bus-device-function format.

For the reasons mentioned above, current address and source identifier extensions, as defined in HT3, do not support the proposed system model.

On the other hand, HyperTransport's scalability limitations may not only be analyzed from the NestID point of view, but also from the interconnect topology perspective. As described in Section 4.1 of the HT3 specification, and using HT3 terminology, HT I/O fabrics are implemented as one or more daisy chains of HT devices, with a bridge to the host system at one end. Multiple daisy chains can be interconnected using bridge devices, forming a tree topology. Additionally, the host can contain multiple bridges, each supporting either a single HT I/O chain or a tree of HT I/O chains.

These topologies were conceived to attach a set of peripheral devices or controllers to a single host. The only exception are double-hosted chains, but even in this case, either one host acts as a slave and routes all of its transactions through the master host, or the chain appears logically as two distinct daisy chains - each attached to only one host bridge. Nevertheless, none of these topologies fit the requirements of large scale systems.

As can be seen, the HT3 specification does not provide suitable support for interconnecting a large number of hosts and for routing messages between them. Also, it does not provide support for identifying nests in the system, as discussed above. Consequently, either the entire specs should be modified to provide the required support, or a clever way to extend the specs while maintaining backward compatibility should be found. Modifying the specs should be ruled out as it would compromise the extensive investments already made by Consortium members in current and previous generations of HT technology. Thus, devising backward compatible HT extensions should be the recommended path to follow. After almost two years of discussions and deliberations among high-level members of the Consortium, the HT extensions have been formalized and released in the form of the High Node Count HyperTransport Specification.

HT Extensions Structure

If HT was to natively support a large number of hosts (or nests), some extensions to the protocol were required. These extensions had to be implemented in such a way that the resulting protocol was as efficient and fast as the current HT3 version. Additionally, the following goals have been considered when extending HT:

1. Analyze the ideal characteristics of extended HT while monitoring HT backward compatibility, so that existing designs could be reused in extended devices as much as possible.
2. Minimize the extensions' overhead - i.e. use of extra bandwidth, additional latency and cost. This should be done without crippling the design.
3. Optimize the extensions by taking into account that the majority of the system platforms will be rather small-scale.
4. Allow for easy addition of new features that could be deemed necessary in the future.

Moreover, the extensions of the HyperTransport protocol had to be done in such a way that support for future features commonly found in large scale systems was straightforward.

As discussed above, interconnecting a large number of hosts requires an interconnect that supports topologies other than current chains and trees, which may not be efficient for interconnecting many hosts. It also requires a global enumeration scheme across multiple hosts, so that each host knows the unique identifier of every other host in the system. Moreover, some efficient routing strategies are needed, especially in those cases where multiple physical paths exist among pairs of hosts. Therefore, in order to build a system with a large number of hosts, the following areas were identified by the Consortium as the minimum extension set to be considered:

- Addressing scheme: ability to address a much larger number of devices. This mainly affects packet formats.
- Network topology: support for topologies with much higher connectivity that will enable many more concurrent transmissions in large platforms, provide shorter paths, and provide alternative paths in case of failure.
- Routing mechanisms: a routing algorithm that supports routing messages in the above topologies. The implementation of the routing logic should be efficient both for small and large systems.

The three issues above are closely related to each other, and design decisions for one of them may significantly impact the others. Additionally, in order to develop an efficient extension of the protocol, the packet format used in the extended HT should be optimized by taking into account - at least - the addressing scheme.

At this point, the HyperTransport Consortium had to make a decision about which features to include in the future extension and how those features would look like. Note that not all such characteristics required to be phased into the new HT specification, as some of them may be implementation-dependent. For example, conveniently defining the packet format (which would necessarily imply defining the addressing scheme) may allow leaving both the network topology and the routing mechanism undefined. These two topics would be addressed/defined by manufacturers when designing their products, or could be considered at a later time for inclusion in subsequent HT specifications. Additionally, this would open up market opportunities for the companies member of the HyperTransport Consortium at the same time that allow these characteristics to mature before developing more advanced specifications. For these reasons, the High Node Count HyperTransport Specification recently released focused on defining the minimum set of extensions to the HT protocol that allow HyperTransport to efficiently support large system sizes. Network topologies and routing mechanisms were classified as implementation-dependent. However, the use of distributed routing is assumed in the interconnect. In this way, packets exchanged between nests will be kept small. Additionally, the amount of routing information contained in them is constant and therefore independent of the path between the source and the destination nests. This simplifies decoding the packets. Moreover, adaptive routing may be used in the future. Finally, other issues, like the required

protocols that set up the identifier of each nest during system initialization were also defined as implementation-dependent.

High Node Count Specification

This section describes and explains the extensions to HT3 intended to allow HyperTransport to natively provide support for high node count environments[1].

One of the topics addressed by the HyperTransport Consortium while defining these extensions was the maximum number of nests to be supported. The addressing scheme and the new packet format depended on this. On one hand, the maximum allowed number of nests should be large enough to support current and future HPC needs. Actually, it should be large enough so that the new specification under development would not require to be modified in the near term. This system size is probably larger than the market may require (at least for quite a while), thus smaller systems would be paying a performance penalty for HT being able to scale up to those large sizes. On the other hand, defining relatively small NestIDs, optimized for smaller, more popular systems, would likely impose the need for further specification extensions in just a few years to satisfy market-driven scalability requirements.

Ultimately, the decision made by the Consortium was to support multiple system sizes. This would keep complexity low for small system configurations while enabling HT to scale up as needed. Of course, it was taken into account the space availability in the format of the extended packets as well as cost and performance constraints.

With the NestID length defined, the format of the new specification extensions was also defined. These extensions are based on a new control doubleword: the NestID extension, which allows nests in the system to identify themselves and also to univocally address other nests, and therefore, it is one of the key elements of the specification extensions.

The format of the new control doubleword allows an easy decoding of the extensions, aligning the new specification with the second goal in the previous section, that established that extensions should keep overhead as low as possible. Additionally, the presence of NestID extensions in a request packet is fully compatible with other extensions such as source identifier or address extensions - remember that the first goal mentioned in the previous section established the need for full backwards compatibility.

Response packets may also be extended by NestID extensions. In fact, when a request reaches a nest, it will probably have to generate a response packet intended for the nest that initially created the request. Therefore, in order to properly forward the response to the right source nest, a NestID extension must be used. Obviously, the DestNest identifier used in a response packet is copied from the SrcNest identifier of the corresponding request packet.

The exact location of the NestID extension in a request packet is not accidental, but designed so to improve system performance because when a nest sends a request to another nest, the DestNest identifier needs to be decoded in order to forward that packet to the target nest. Therefore, properly locating the required information in the request packet reduces routing time. Additionally, the location

[1] It should be noticed that the HT Consortium has decided not to make the new extensions available to the general public, so that they will only be usable by HT Consortium Promoter and Contributor members. Therefore, because of confidentiality constraints, the description in this section will not provide the complete details of the new extensions.

of the DestNest identifier in response packets has been carefully designed so that not only routing time is minimized but also routing of both request and response packets in the fabric is kept efficient.

Moreover, packet overhead is minimized for small systems. The NestID extension has been meticulously designed for small-scale systems, where the new format is heavily optimized. Actually, this was the third goal to keep in mind mentioned in the previous section. In small systems, packet length optimization translates in request packets being 20% shorter than non-optimized packets. Additionally, they are 42% shorter if compared with packets for large systems.

Note that some extra configuration information is needed in a nest so that it properly interprets the new extensions. The required configuration information is located in the High Node Count Capability Block, the other master piece of the specification. Unfortunately, because of confidentiality constraints, the format and usage of this capability block cannot be disclosed.

Protocol Extensions Implementation

The High Node Count HyperTransport Specification can be implemented in two different ways, each of them having a different impact on cost, benefits, and market opportunities. The following discussion is independent of the maximum global memory size and thus it is equally valid for any of the choices mentioned above.

Native Implementation

The implementation option yielding the best system performance would be modifying the HT logic within the Opteron processors, in order to enable them to inter-communicate directly using the proposed extension and to avoid the use of any external logic, thereby minimizing latency and maximizing bandwidth. Additionally, the embedding of such HT extensions into Opteron processors interconnected by large inter-processor networks would require embedding in the Opteron architecture some kind of routing logic also, so that HT packets are forwarded from the given source to the proper destination. This approach would be similar to the Alpha 21364 processor [6], which integrated a router function and allowed processors to be interconnected by a 2D torus with a maximum of 128 processors. Figure 5 shows a similar scenario of a 2D mesh with 16 processors. In this case, processors would require only four HT links for complete interconnection. If more links are available - e.g. by means of HT3 link-splitting capability - more efficient topologies like 3D meshes or tori could be deployed. These topologies require 6 links per processor. This implementation option is certainly the most effective long-term for best performance and lowest implementation cost. However, it is not the most ideal time-to-market wise and cost wise, as modifying the processor's logic would be quite time and resource laden.

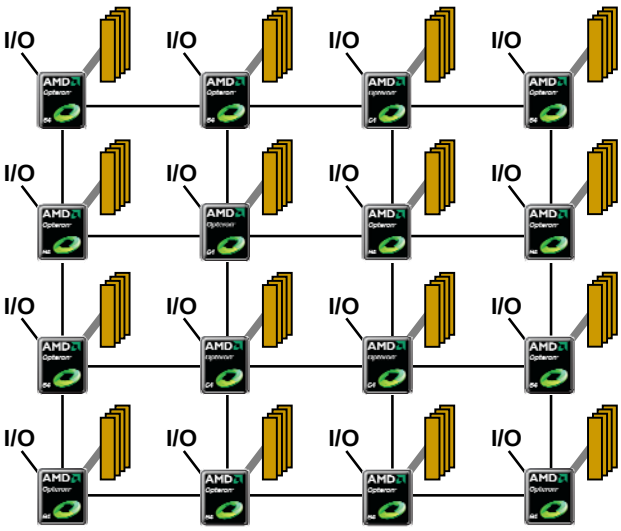

Figure 5: 16-way 2D-mesh system. Processors have embedded protocol extension and routing logic

Bridged Implementation

Actually, embedding these new HT extensions does not necessarily impose changes to the processor design. In fact, an alternative, more flexible and time-to-market friendly choice would be the implementation of the extended HT functionality in external logic - i.e. in a bridge chip - whose main purpose would be translating requests and responses from one version of the protocol to the other - i.e. a chip that implements standard HT3 on one side and extended HT on the other. The logic inside this chip could be designed in such a way that it detects a current HT chain on one side and a large HT network on the other. The chip should also incorporate some routing capabilities in order to forward messages to their intended destination. Figure 6 shows such a system, composed of 16 processors that communicate with each other through such bridge chips.

The technical and commercial advantages of such external logic solution greatly outweigh its disadvantages compared to the first option. The bridge implementation would allow HT extensions' functionality to mature before full integration into the CPU. It would also open up interesting market opportunities for HyperTransport Consortium member companies and the HT technology ecosystem. The trade-off will be a slightly higher latency and higher system cost (1 bridge chip per nest).

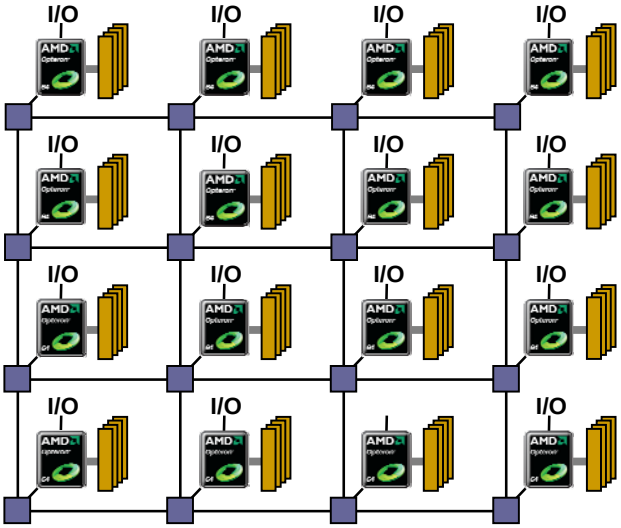

Figure 6: 16-way 2D-mesh system. Protocol extension and routing logic integrated in the bridge chip

The implementation of the bridge chip requires including a matching unit that associates returning responses to previously sent requests. In the case for the bridge chip at the source end, it must first extend the original request by using the NestID extensions. Once the extended packet arrives at the destination bridge chip, it must aggregate requests from several sources by translating them into local requests which are uniquely defined by the combination of UnitID and SrcTag. Thus, the destination bridge must store the value of the UnitID, SrcTag, and SrcNest fields in the incoming packet along with the new local values for UnitID and SrcTag, so that when the response comes from the local chain, the bridge can associate it to the initially received request and translate back those fields to the initial ones before sending back the response to the source end. Once the response is received at the source end, the source bridge will remove the NestID extensions to deliver a final response to the initial source of the request.

Conclusions

Large computing systems are interesting because of their aggregate computing power and overall memory capacity. These large systems require high bandwidth, low latency interconnect technologies for inter-processor communication. HT3 has the capability and latitude required by these systems, except for its inability to scale appropriately. Fortunately, it is feasible and cost-effective to infuse such needed scalability into HyperTransport, as described by the High Node Count HyperTransport Specification.

The bridged implementation of such specification may be quick-to-market, not requiring changes to current processor architectures, but will not provide the best latency performance. On the contrary, the integration of the new HT extensions in the processor architecture would allow future Opteron chips to be powerful building blocks for systems of any viable size and scale.

The proposed protocol extensions are fully compatible with the HT3 specification. Packet ordering is preserved by processing packets in compliance with current ordering rules. On the other hand, flow control is also complied with because extended packets will use current buffers and therefore no change is required in the NOP flow control packets. Nevertheless, buffer size should be enlarged in order to store the extended packets.

Regarding protocol overhead, the proposed extensions add a few bytes to current HT packets, but only in the case of packets targeted to remote processors. Instead, packets travelling through the local HT chain and intended for local I/O do not require to be extended and, therefore, no overhead is added. Moreover, for those cases where overhead is a primary concern, the new protocol extension has been optimized for smaller system sizes.

References

[1] A. Ahmed, P. Conway, B. Hughes, and F. Weber, “Hammer Shared Memory Multi Processor Systems”, HotCHips 14, August 2002

[2] W. Camp, "Computer Architecture: Opportunities and Challenges for Scalable Applications", Sandia CSRI Workshop on Next-generation scalable applications: When MPI-only is not enough, June 2008

[3] E. Chow, “Non-MPI Apps: Why we don't use MPI-only”, Sandia CSRI Workshop on Next-generation scalable applications: When MPI-only is not enough, June 2008

[4] Cray Inc., “Cray XT5 Specifications”, available at http://www.cray.com/Products/XT/Product/Specifications.aspx, 2008

[5] HyperTransport Technology Consortium, “HyperTransport I/O Link Specification Revision 3.10”, available at http://www.hypertransport.org, 2008

[6] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The Alpha 21364 Network Architecture", IEEE Micro, 22:26-35 January, 2002

[7] R. Oehler and R. Kota, “HORUS - Enabling large scale, 32-way Opteron Enterprise Servers”, HotCHips 16, August 2004

[8] J. Young, S. Yalamanchili, F. Silla, and J. Duato, "A HyperTransport-Enabled Global Memory Model for Improved Memory Efficiency”, in proceedings of First International WorkShop on HyperTransport Research and Applications, February 2009

[9] K. Yelick, "Programming Models: Opportunities and Challenges for Scalable Applications", Sandia CSRI Workshop on Next-generation scalable applications: When MPI-only is not enough, June 2008